Damas-Milner Type System: type inference and term generation.

Definition 1. *Generalization*

$$\boldsymbol{G}(\tau, E) \;=\; \forall \alpha_1, \ldots, \alpha_n . \tau$$

where $\alpha_1, \ldots \alpha_n$ are exactly these free variables of τ, which do not occur in E.

If $E = [x_1 \colon \tau_1; x_2 \colon \tau_2; \ldots; x_k \colon \tau_k]$, and $1 \leqslant n \leqslant k$ then we write $E(n) = x_n \colon \tau_n$ and $E(x_n) = \tau_n$. For length we write $\bar{E} = k$.

Definition 2. *Specialization of a type for a variable from environment: τ is a specialization of a type of variable x from environment E*

$$\tau \leqslant E(x) \;\equiv\; E = \ldots; x \colon \forall \alpha_1 \ldots \alpha_n . \sigma; \ldots \text{ and } \tau = [\tau_1 / \alpha_1; \ldots; \tau_n / \alpha_n] \sigma$$

for some $\tau_1, \ldots, \tau_n$. A type σ is more general than τ

$$\tau \leqslant \sigma \;\equiv\; \tau = \forall \gamma_1 \ldots \gamma_m . \tau' \wedge \sigma = \forall \alpha_1 \ldots \alpha_n . \sigma' \wedge \tau' = [\tau_1 / \alpha_1; \ldots; \tau_n / \alpha_n] \sigma'$$

for some $\tau_1, \ldots \tau_n$. Substitution $T = [\tau_1 / \alpha_1; \ldots; \tau_n / \alpha_n]$ denotes replacement of free occurrences of variables α_i by corresponding types τ_i.

Definition 3. *Typing rules for the Mini-ML language:*

1. *VAR: Variables.*

$$\frac{\tau \leqslant E(x)}{E \vdash x \colon \tau}$$

2. *FIX: Functions.*

$$\frac{E.f \colon \sigma \rightarrow \tau . x \colon \sigma \vdash e \colon \tau}{E \vdash (\mathrm{fix}\, f x . e) \colon \sigma \rightarrow \tau}$$

3. *APP: Applications.*

$$\frac{E \vdash e \colon \sigma \rightarrow \tau \qquad E \vdash e' \colon \sigma}{E \vdash e e' \colon \tau}$$

4. *LET: Local bindings.*

$$\frac{E \vdash e' \colon \sigma \qquad E . x \colon \boldsymbol{G}(\sigma, E) \vdash e \colon \tau}{E \vdash (\mathrm{let}\, x = e' \,\mathrm{in}\, \mathrm{e}) \colon \tau}$$

Lemma 4. *When $E \vdash \tau$ and type constants $d_1, \ldots, d_n$ do not occur in E, then*

$$E \vdash \forall \alpha_1 \ldots \alpha_n . [\alpha_1 / d_1; \ldots; \alpha_n / d_n] \tau$$

Type inference algorithm $\mathcal{W}$ and term generation algorithm $\mathcal{C}$.

Definition 5. $\mathcal{W}(E, e, V) = (T, \tau, V')$*, where match e with:*

1. *$e = x$ (VAR: Variables). $T = \varnothing$ and for $x \colon \forall \alpha_1 \ldots \alpha_n . \sigma \in E$ let*

$$\begin{aligned} \tau &= [\vec{\beta} / \vec{\alpha}] \sigma \\ V' &= V' \setminus \vec{\beta} \\ T &= \varnothing \end{aligned}$$

2. $e = \text{fix}\, fx.e_1$ *(FIX: Functions). Let*

$$\begin{aligned} V &= \{\beta, \beta_1\} \dot\cup V'' \\ (R, \rho, V') &= \mathcal{W}(E.f{:}\, \beta_1 \to \beta.x{:}\, \beta_1, e_1, V'') \\ U &= \boldsymbol{U}(R\beta, \rho) \\ T &= UR \\ \tau &= UR(\beta_1 \to \beta) \end{aligned}$$

3. $e = f\,g$ *(APP: Applications). Let*

$$\begin{aligned} (R, \rho, V_1) &= \mathcal{W}(E, f, V) \\ (S, \sigma, V_2) &= \mathcal{W}(RE, g, V_1) \\ U &= \boldsymbol{U}(S\rho, \sigma \to \beta) \\ V' &= V_2 \setminus \{\beta\} \\ T &= USR \\ \tau &= U\beta \end{aligned}$$

4. $e = \text{let}\, x = f \,\text{in}\, g$ *(LET: Local bindings). Let*

$$\begin{aligned} (R, \rho, V_1) &= \mathcal{W}(E, f, V) \\ (S, \sigma, V') &= \mathcal{W}(RE.x{:}\, \boldsymbol{G}(\rho, RE), g, V_1) \\ T &= SR \\ \tau &= \sigma \end{aligned}$$

Definition 6. $\mathcal{C}(E, \tau, V, \vec{w}) = (T, e, V', \vec{w}\,')$, *where for* $(w_1 \bmod 4)$ *equal*

1. *VAR: Variables. For* $E(w_2 \bmod \bar{E}) = x{:}\, \forall\alpha_1 \ldots \alpha_n.\sigma$ *let*

$$\begin{aligned} U &= \boldsymbol{U}(\tau, [\beta_1/\alpha_1] \ldots [\beta_n/\alpha_n]\sigma) \\ V &= \{\beta_i | 1 \leqslant i \leqslant n\} \dot\cup V' \\ e &= x \\ T &= U \\ w_i' &= w_{i+2} \end{aligned}$$

If the unifier does not exist, $\mathcal{C}$ *is undefined.*

2. *FIX: Functions. If* $\tau = \sigma \to \rho$, *let*

$$\begin{aligned} (R, e_1, V', \vec{w}_1) &= \mathcal{C}(E.f{:}\, \tau.x{:}\, \sigma, \rho, V, (w_{i+1})) \\ T &= R \\ e &= \text{fix}\, fx.e_1 \\ \vec{w}\,' &= \vec{w}_1 \end{aligned}$$

where f, x *are new variables. If* $\tau = \alpha$, *where* α *is a type variable, for* $\beta_1, \beta \in V$ *let*

$$\begin{aligned} R &= [\beta_1 \to \beta/\alpha] \\ (S, e_1, V', \vec{w}_1) &= \mathcal{C}(RE.f{:}\, \beta_1 \to \beta.x{:}\, \beta_1, \beta, V \setminus \{\beta_1, \beta\}, (w_{i+1})) \\ T &= SR \\ e &= \text{fix}\, fx.e_1 \end{aligned}$$

where f, x *are new variables. If* τ *has a different form,* $\mathcal{C}$ *is undefined.*

3. APP: Applications. For $\beta \in V$ let

$$\begin{aligned}
(R, f, V_1, \vec{w}_1) &= \mathcal{C}(E, \beta \to \tau, V \setminus \{\beta\}, (w_{i+1}))\\
(S, g, V', \vec{w}_2) &= \mathcal{C}(R E, R\beta, V_1, \vec{w}_1)\\
T &= S R\\
e &= f g\\
\vec{w}\,' &= \vec{w}_2
\end{aligned}$$

4. LET: Local bindings. For $\beta \in V$ let

$$\begin{aligned}
(R, f, V_1, \vec{w}_1) &= \mathcal{C}(E, \beta, V \setminus \{\beta\}, (w_{i+1}))\\
(S, g, V', \vec{w}_2) &= \mathcal{C}(R E.x\colon \boldsymbol{G}(R\beta, R E), R\tau, V_1, \vec{w}_1)\\
T &= S R\\
e &= \mathrm{let}\, x = f \,\mathrm{in}\, g\\
\vec{w}\,' &= \vec{w}_2
\end{aligned}$$

Soundness.

Theorem 7. *Let e be an expression, E an environment, V a set of type variables. If $(T, \tau, V') = \mathcal{W}(E, e, V)$ is defined, then we can derive $T E \vdash e\colon \tau$.*

Theorem 8. *Let τ be a type, E an environment, V a set of type variables. If $(T, e, V') = \mathcal{C}(E, \tau, V)$ is defined, we can derive $T E \vdash e\colon T\tau$.*

Completeness.

Theorem 9. *Let e be an expression, E an environment, V an infinite set of variables such, that $V \cap \boldsymbol{F}(E) = \varnothing$. If there exists type τ' and substitution T' such that $T'E \vdash e\colon \tau'$, then $(T, \tau, V') = \mathcal{W}(E, e, V)$ is defined, and there exists a substitution P such that*

$$\tau' = P\tau \qquad \textit{and} \qquad T' = PT \textit{ outside } V$$

Definition 10. *Type σ is more general than τ*

$$\tau \preccurlyeq \sigma \equiv \textit{ there exists substitution } P\colon P\sigma = \tau$$

Substitution S is more general than R

$$\begin{aligned}
R \preccurlyeq S &\equiv \textit{ there exists substitution } P\colon PS = R\\
&\equiv \mathrm{Dom}(S) \subseteq \mathrm{Dom}(R) \textit{ i } \forall \alpha \in \mathrm{Dom}(S)\colon R\alpha \preccurlyeq S\alpha
\end{aligned}$$

Theorem 11. *Let e be an expression, E an environment. For any type τ and substitution T' such that $T'E \vdash e\colon T'\tau$ and $\boldsymbol{F}(T') \cap \mathrm{Dom}(T') = \varnothing$, for infinite set of variables V disjoint with $\boldsymbol{F}(E)$ and $\boldsymbol{F}(T')$, for some path of choices the following holds: $(T, e, V') = \mathcal{C}(E, \tau, V)$ and $T' \preccurlyeq T$ outside V.*

Corollary 12. *If β is a type variable not occurring in E, $\beta \notin V$, and $\mathcal{C}(E, \beta, V) = (T, e, V_1)$, then for any type τ and substitution T', $\beta \notin \boldsymbol{F}(\tau) \cup \boldsymbol{F}(T')$, for which $T'E \vdash e\colon \tau$, we have $T' \preccurlyeq T$ outside $V \cup \{\beta\}$ i $\tau \preccurlyeq T\beta$.*

Extending the language with the construct case.

Definition 13. *Typing rules for the inductive structures:*

1. *VAR: Variables...*
2. *FIX: Functions...*
3. *APP: Applications...*
4. *LET: Local bindings...*
5. *CASE: Deconstruction*

$$\frac{E \vdash e' : d\vec{\tau} \qquad E \vdash e_i : \mathrm{Inst}_{c_i}\vec{\tau} \to \theta \, (1 \leqslant i \leqslant n)}{E \vdash \mathrm{case}\, e' \,\mathrm{of}\, \{c_1 \Rightarrow e_1 | ... | c_n \Rightarrow e_n\} : \theta} \; \textit{if } \boldsymbol{C}(d) = \{c_1, ..., c_n\}$$

6. *Instead of the rule: CONS: Constructor*

$$\frac{}{E \vdash c : \mathrm{Inst}_c\vec{\tau} \to d\vec{\tau}} \; \textit{if } c \in \boldsymbol{C}(d)$$

we introduce an assumption on environment E:

$$(\forall d)(\forall c \in \boldsymbol{C}(d)) \;\; \textit{let } c : \forall \vec{\alpha}.\mathrm{Inst}_c\vec{\alpha} \to d\vec{\alpha} \in E, \;\; \textit{where } \#\vec{\alpha} \textit{ is the arity of } d$$

where $\boldsymbol{C}(d)$ *is a set of constructors of the type* d*,* $\vec{\tau}$ *are parameters of the type,* $\mathrm{Inst}_c\vec{\tau}$ *is a vector of argument types of constructor* c*, when the parametric type* d *takes parameters* $\vec{\tau}$*. Notation: if* $\vec{\sigma} = (\sigma_1, ..., \sigma_n) = (\sigma_i)_{1 \leqslant i \leqslant n}$*, then* $\#\vec{\sigma} = n$ *and*

$$\vec{\sigma} \to \tau = \sigma_1 \to ... \to \sigma_n \to \tau = \sigma_1 \to (... \to (\sigma_n \to \tau)...)$$

Definition 14. $\mathcal{W}(E, e, V) = (T, \tau, V')$*, where match* e *with:*

1. $e = x$ *(VAR: Variables)...*
2. $e = \mathrm{fix}\, fx.e_1$ *(FIX: Functions)...*
3. $e = f g$ *(APP: Applications)...*
4. $e = \mathrm{let}\, x = f \,\mathrm{in}\, g$ *(LET: Local bindings)...*
5. $e = \mathrm{case}\, e' \,\mathrm{of}\, \{c_1 \Rightarrow e_1 | ... | c_n \Rightarrow e_n\}$ *(CASE: Deconstruction) For type* d *with arity* k *(k-parametric) such that* $c_1, ..., c_n \in \boldsymbol{C}(d)$

$$\begin{array}{rcl}
V & = & \{\beta_i | 1 \leqslant i \leqslant k\} \cup \{\theta\} \dot{\cup} V_r \\
(T_0, \rho, V_0) & = & \mathcal{W}(E, e', V_r) \\
U_0 & = & \boldsymbol{U}(\rho, d\vec{\beta}) \\
(T_1, \sigma_1, V_1) & = & \mathcal{W}(U_0 T_0 E, e_1, V_0) \\
U_1 & = & \boldsymbol{U}(T_1 U_0 T_0 (\mathrm{Inst}_{c_1}\vec{\beta} \to \theta), \sigma_1) \\
... & & \\
(T_n, \sigma_n, V_n) & = & \mathcal{W}(U_{n-1} T_{n-1} ... U_0 T_0 E, e_n, V_{n-1}) \\
U_n & = & \boldsymbol{U}(T_n U_{n-1} T_{n-1} ... U_0 T_0 (\mathrm{Inst}_{c_1}\vec{\beta} \to \theta), \sigma_n) \\
T & = & U_n T_n ... U_0 T_0 \\
\tau & = & T\theta \\
V' & = & V_n
\end{array}$$

Definition 15. *$\mathcal{C}(E,\tau,V,\vec{w})=(T,e,V',\vec{w}\,')$, where for $(w_1 \bmod 5)$ equal*

1. *VAR: Variables...*

2. *FIX: Functions...*

3. *APP: Applications...*

4. *LET: Local bindings...*

5. *CASE: Deconstruction. Let d be $(w_{i+1} \bmod D)$-th inductive type, where D is the number of inductive types, and k is the arity of type d. Let*

$$\begin{array}{rcl} V & = & \{\beta_i | 1 \leqslant i \leqslant k\} \cup \{\theta\} \dot{\cup} V_r \\ (R, e', V_0, \vec{w}_0) & = & \mathcal{C}(E, d\vec{\beta}, V_r, (w_{i+2})) \\ (T_1, e_1, V_1, \vec{w}_1) & = & \mathcal{C}(R\,E, R\,(\mathrm{Inst}_{c_1}\vec{\beta} \rightarrow \tau), V_0, \vec{w}_0) \\ \ldots & & \\ (T_n, e_n, V_n, \vec{w}_n) & = & \mathcal{C}(T_{n-1} \ldots T_1 R\,E, T_{n-1} \ldots T_1 R\,(\mathrm{Inst}_{c_n}\vec{\beta} \rightarrow \tau), V_{n-1}, \vec{w}_{n-1}) \\ T & = & T_n \ldots T_1 R \\ e & = & \mathrm{case}\, e'\, \mathrm{of}\, \{c_1 \Rightarrow e_1 | \ldots | c_n \Rightarrow e_n\} \\ V' & = & V_n \\ \vec{w}\,' & = & \vec{w}_n \end{array}$$

Soundness with case.

Theorem 16. *Let e be an expression, E an environment, V a set of type variables. If $(T, \tau, V')=\mathcal{W}(E,e,V)$ is defined, then we can derive $T E \vdash e{:}\, \tau$.*

Theorem 17. *Let τ be a type, E an environment, V a set of type variables. If $(T,e,V')=\mathcal{C}(E,\tau,V)$ is defined, we can derive $T E \vdash e{:}\, T\tau$.*

Completeness with case.

Theorem 18. *Let e be an expression, E an environment, V an infinite set of type variables such that $V \cap \boldsymbol{F}(E) = \varnothing$. If there exists a type τ' and a substitution T' such that $T'E \vdash e{:}\, \tau'$, then $(T,\, \tau,\, V') = \mathcal{W}(E,\, e,\, V)$ is defined, and there exists a substituion P such that*

$$\tau' = P\tau \qquad i \qquad T' = PT \textit{ outside } V$$

Theorem 19. *Let e be an expression, E an environment. For any type τ and substitution T' such that $T'E \vdash e{:}\, T'\tau$ and $\boldsymbol{F}(T') \cap \mathrm{Dom}(T') = \varnothing$ (?), for an infinite set of variables V disjoint with $\boldsymbol{F}(E)$ and $\boldsymbol{F}(T')$, for some path of choices the following holds: $(T,\, e,\, V') = \mathcal{C}(E,\tau,V)$ and $T' \preccurlyeq T$ outside V.*